Avoiding Moving Obstacles during Visual Navigation

Andrea Cherubini, Boris Grechanichenko, Fabien Spindler and François Chaumette

***Abstract*— Moving obstacle avoidance is a fundamental requirement for any robot operating in real environments, where pedestrians, bicycles and cars are present. In this work, we design and validate a new approach that takes explicitly into account obstacle velocities, to achieve safe visual navigation in outdoor scenarios. A wheeled vehicle, equipped with an actuated pinhole camera and with a lidar, must follow a path represented by key images, without colliding with the obstacles. To estimate the obstacle velocities, we design a Kalman-based observer. Then, we adapt the tentacles designed in [1], to take into account the predicted obstacle positions. Finally, we validate our approach in a series of simulated and real experiments, showing that when the obstacle velocities are considered, the robot behaviour is safer, smoother, and faster than when it is not.**

***Index Terms*— Visual Servoing, Visual Navigation, Collision Avoidance.**

I. INTRODUCTION

Autonomous driving has become a prominent application domain for robotics research, as witnessed by a cornucopia of publications in this area [2 - 4]. The success of the DARPA Urban Challenges [5] has heightened expectations that autonomous cars will soon be able to operate in environments of realistic complexity. Nevertheless, robot motion safety remains a critical issue. In this context, a fundamental research field is obstacle avoidance, which has traditionally been handled by two techniques [6]: the deliberative approach, usually consisting of a motion planner, and the reactive approach, based on the instantaneous sensed information. In the works that we will cite below, the obstacle velocities have also been considered in the avoidance method.

The approach presented in [7] is one of the first where static and moving obstacles are avoided, based on their current positions and velocities relative to the robot. The manoeuvres are generated by selecting robot velocities outside of the *velocity obstacles*, that would provoke a collision at some future time. This paradigm has been adapted in [8] to the car-like robot kinematic model, and extended in [9] to take into account unpredictably moving obstacles. Another pioneer method that has inspired many others is the Dynamic Window [10], that is derived directly from the dynamics of the robot, and is especially designed to deal with constrained velocities and accelerations. A generalization of the dynamic map that accounts for moving obstacle velocities and shapes is presented in [11], which utilizes a union of polygonal zones corresponding to the non admissible velocities. In [12], the Dynamic Window has been integrated in a graph search algorithm for path planning, to drive the robot trajectories. A planning approach is also used in [13], where the likelihood of obstacle positions is input to a Rapidly-exploring Random Tree algorithm. In [14], motion safety is characterized by three criteria, respectively related to the model of the robotic system, to the model of the environment and to the decision making process. The author proves that motion safety cannot be guaranteed in the presence of moving objects (i.e., the robot may inevitably collide at some time in the future). More recently [15], the same researchers have defined the Braking Inevitable Collision States as states such that, whatever the future braking trajectory, a collision will occur.

A. Cherubini and B. Grechanichenko are with the Laboratory for Computer Science, Microelectronics and Robotics LIRMM - Université de Montpellier 2 CNRS, 161 Rue Ada, 34392 Montpellier, France. {firstname.lastname}@lirmm.fr.
F. Spindler and F. Chaumette are with INRIA Rennes - Bretagne Atlantique, IRISA. {firstname.lastname}@inria.fr

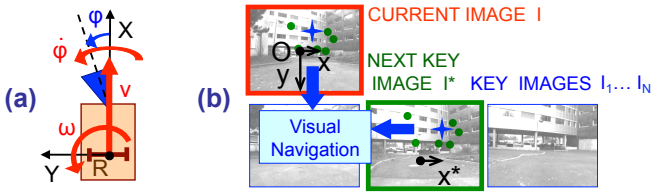

Fig. 1. General definitions. (a) Top view of the robot (orange), with actuated camera (blue). (b) Current and next key images, and key image database.

Although all these approaches have proved effective, none of them deals with moving obstacle avoidance during visual navigation. In [1], we presented a framework that guarantees that obstacle avoidance has no effect on a visual task. In the present paper, we further improve that framework, by designing a reactive approach that can deal with moving obstacles as well. A wheeled vehicle, equipped with an actuated pinhole camera and with a forward-looking lidar, must follow a path represented by key images, without colliding. Our approach is based on tentacles [16], i.e. candidate trajectories (arcs of circles) that are evaluated during navigation, both for assessing the context, and for designing the task in case of danger. The main contribution of this paper is the improvement of that framework, to take into account the obstacle velocities. We have designed a Kalman-based observer for estimating the obstacle velocities, and then adapted the tentacles designed in [1], to effectively take into account these velocities. Our approach is validated in a series of experiments.

The article is organized as follows. In Sect. II, all the relevant variables are defined. In Section III and IV, we explain respectively how the obstacle velocities are estimated, and how they are used to predict possible collisions. Then, in Sect. V, the control law from [1] is recalled, and adapted to deal with moving obstacles. Experimental results are reported in Section VI, and summarized in the Conclusion.

II. PROBLEM DEFINITION

This section is, in part, taken from [1]. Referring to Fig. 1, we define the robot frame $\mathcal{F}_{\mathcal{R}}\,(R, X, Y)$ (R is the robot center of rotation) and image frame $\mathcal{F}_{\mathcal{I}}(O, x, y)$ (O is the image center). The robot control inputs are $\mathbf{u} = [v,\ \omega,\ \dot{\varphi}]^{\top}$. These are the translational and angular velocities of the vehicle, and the camera pan angular velocity. We use the normalized perspective camera model, and we assume that the sequence of images that defines the path can be tracked with continuous $v\,(t) > 0$. This ensures safety, since only obstacles in front of the robot can be detected by our scanner.

The path that the robot must follow is represented as a database of ordered key images, such that successive pairs

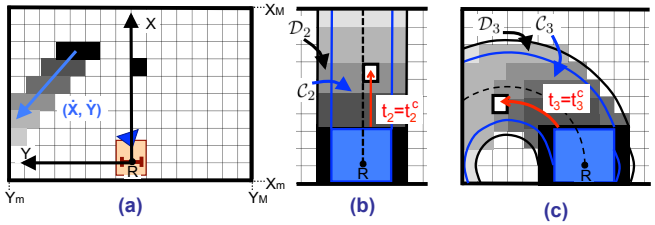

Fig. 2. Obstacle models. (a) A static (right) and moving (left) object are observed (black); we show the object velocity in cyan, and future occupied cells c_i in grey, increasingly light with t_{i0}. (b, c) Tentacles (dashed black), with classification areas (collision in blue, dangerous in black), corresponding boxes and delimiting arcs of circle, and cells $\mathbf{c}_i \in \mathcal{D}_j$ displayed in grey, increasingly light with increasing t_{ij}.

contain some common static visual features (points). First, the vehicle is manually driven along a *taught* path, with the camera pointing forward ($\varphi = 0$), and all the images are saved. Afterwards, a subset (database) of N key images $I_1, \ldots, I_N$ representing the path (Fig. 1(b)) is selected. Then, during autonomous navigation, the current image, noted I, is compared with the next key image $I^* \in \{I_1, \ldots, I_N\}$, and a relative pose estimation between I and I^* is used to check when the robot passes the pose where I^* was acquired. For key image selection, and visual point detection and tracking, we use the algorithm in [17]. The output of this algorithm, which is used by our controller, is the set of points visible both in I and I^*. Then, navigation consists of driving the robot forward, while I is driven to I^*. We maximize similarity between I and I^* using only the abscissa x of the centroid of points matched on I and I^* to control the robot heading. When I^* has been passed, the next image in the set becomes the desired one, and so on, until I_N is reached.

Along with the visual path following problem, we consider obstacles which are on the path, but not in the database, and sensed by the lidar in a plane parallel to the ground. For obstacle modeling, we use the occupancy grid in Fig. 2(a): it is linked to $\mathcal{F}_\mathcal{R}$, with cell sides parallel to X and Y. Its extension is limited ($X_m \leq X \leq X_M$ and $Y_m \leq Y \leq Y_M$), to ignore obstacles that are too far to jeopardize the robot. Any grid cell $\mathbf{c} = [X, Y]^\top$ is considered currently occupied (black in Fig. 2(a)) if an obstacle has been sensed there. For cells lying in the scanner area, only the current scanner reading is considered. For the other cells, we use past readings, displaced with odometry. The set of occupied cells with their estimated velocities, denoted by $\mathcal{O}$, is used, along with the robot geometric and kinematic characteristics, to derive possible future collisions. This approach is different from the one in [1], where only the *currently* occupied cells in the grid were considered. To estimate the obstacle velocities, and therefore update $\mathcal{O}$, we have designed an obstacle observer, detailed below.

III. OBSTACLE OBSERVER

The detection and tracking of objects is crucial for collision-free navigation. Of particular interest are potentially dynamic objects (i.e., objects that could move) since their presence and potential change of state will influence the planning of actions and trajectories. Obviously, estimating the velocity of these objects is fundamental.

Compared to areas where known road network information can provide background separation, unknown environments present a more challenging scenario, due to low signal to noise ratio. Recent works [18], [19] have tackled these issues. In [18], classes of interest for autonomous driving (i.e., cars, pedestrians and bicycles) are identified, using shape information and a RANSAC-based edge selection algorithm. The authors of [19] apply a foreground model that incorporates geometric as well as temporal cues; then, moving vehicles are tracked using a particle filter. Both works rely on the Velodyne HDL-64E S2, a laser range finder that provides rich 3D point clouds, to classify moving obstacles. Instead, we target solutions based uniquely on a 2D lidar, and we are not interested in recognizing the object classes.

In practice, we base our work on two assumptions. First, we consider all objects to be rigid (this is plausible for the projection on the ground of walls, most vehicles and even pedestrians). Second, we consider the instantaneous curvature of their trajectories (i.e., the ratio between their angular and translational velocities) small enough to assume that their motion is purely translational over short time intervals. Hence, the translational velocities of all points on an object are identical and equal to that of its centroid.

Then, for each object, the state to be estimated will be composed of the coordinates of its centroid in $\mathcal{F}_\mathcal{R}$, and by their derivatives:

$$\mathbf{x} = \left[X, Y, \dot{X}, \dot{Y}\right]^\top.$$

Using a first-order Markov model (which is plausible for low object accelerations), the state at time t is evolved from the state at $t - \Delta t$ (Δt is the sampling time) according to:

$$\mathbf{x}(t) = \mathbf{F}\mathbf{x}(t - \Delta t) + \mathbf{w}(t), \tag{1}$$

where $\mathbf{w}(t) \sim N(0, \mathbf{Q})$ is assumed to be Gaussian white noise, with covariance $\mathbf{Q}$ and, the state transition model is:

$$\mathbf{F} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}.$$

At time t, an observation $\mathbf{z}(t)$ of the object centroid coordinates is derived from scanner data. It is related to the state by:

$$\mathbf{z}(t) = \mathbf{H}\mathbf{x}(t) + \mathbf{v}(t), \tag{2}$$

where $\mathbf{v}(t) \sim N(0, \mathbf{R})$ is assumed to be Gaussian white noise with covariance $\mathbf{R}$, and the observation model is:

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}.$$

Let us outline the steps of our recursive algorithm for deriving $\hat{\mathbf{x}}(t)$ (our estimate of $\mathbf{x}(t)$), based on current observations $\mathbf{z}(t)$, and on previous states $\mathbf{x}(t - \Delta t)$.

1) At time t, all currently occupied cells in $\mathcal{O}$ are clustered in objects, using a threshold on pairwise cell distance, and the current observation of the centroid coordinates $\mathbf{z}(t)$ is derived for each object.
2) All of the object centroids that have been observed at some time in the recent past (we look back in the last $2s$) are displaced by odometry, to derive their coordinates $X(t - \Delta t)$ and $Y(t - \Delta t)$.
3) The observed and previous object centroids (outputs of steps 1 and 2) are pairwise matched according to their distance. We then discern between three cases:
   - For matched objects, the previous centroid velocity is obtained by numerical differentiation:

$$\begin{cases} \dot{X}(t - \Delta t) = (z_1(t) - X(t - \Delta t))/\Delta t \\ \dot{Y}(t - \Delta t) = (z_2(t) - Y(t - \Delta t))/\Delta t \end{cases}.$$

These complete, along with the outputs of step 2, the state vector $\mathbf{x}(t-\Delta t)$. A Kalman filter can then be applied to equations (1) and (2), to derive $\hat{\mathbf{x}}(t)$.

- For unmatched objects currently observed, we set:

$$\hat{\mathbf{x}}(t) = \begin{bmatrix} \mathbf{z}(t) \\ 0 \\ 0 \end{bmatrix}.$$

- For unmatched unobserved objects, the centroid coordinates are memorized (these will be the inputs for step 2).

The output of our algorithm is, at each iteration t, the estimate of the object centroid coordinates and of its velocities:

$$\hat{\mathbf{x}}(t) = \left[\hat{X}(t), \hat{Y}(t), \hat{\dot{X}}(t), \hat{\dot{Y}}(t)\right]^{\top}.$$

Then, each currently occupied cell $\mathbf{c}_i$ is associated to the estimated velocity of the object it belongs to, or to null velocity, if it has not been associated to any object. Set $\mathcal{O}$ is finally formed by all the occupied cell states:

$$\begin{bmatrix} \mathbf{c}_i \\ \dot{\mathbf{c}}_i \end{bmatrix}^{\top} \in \mathcal{O},$$

that encode the cell current coordinates and velocities in the robot frame. In the next section, we will show how $\mathcal{O}$ is used to predict possible collisions, and accordingly adapt the control strategy. We will assess the danger of each cell by considering the time that the robot will navigate before eventually colliding with it. Without loss of generality, in the next section this time is measured from the current instant t.

IV. OBSTACLE MODELLING

A. Obstacle occupation times

At this stage, the trajectory of each occupied cell in $\mathcal{O}$ can be predicted to evaluate possible collisions with the robot. More concretely, we will just estimate the times at which each cell in the grid will be - eventually - occupied by an obstacle. We assume that velocities of all occupied cells in $\mathcal{O}$ remain constant over time horizon T. Then, for each $\mathbf{c}_i$ that may be occupied by an obstacle within T, we can predict initial

$$t_{i0}(\mathbf{c}_i, \mathcal{O}) \in [0, T]$$

and final

$$t_{if}(\mathbf{c}_i, \mathcal{O}) \in [t_{i0}, T]$$

obstacle occupation times, as a function of the set of occupied cell states $\mathcal{O}$. For cells occupied by a static object and belonging to $\mathcal{O}$, we obtain $t_0 = 0$ and $t_f = T$. For cells that will not be occupied within time T, we set $t_0 = t_f = \infty$. Examples of a static (1 cell) and moving (3 cells) object are shown in Fig. 2(a), with future occupied cells c_i displayed in grey, increasingly light with increasing t_{i0}. Below, we explain how the cell occupation times t_0 and t_f will be used to check collisions with the possible robot trajectories.

B. Tentacles

As in [1], we use a set of drivable paths (tentacles), both for perception and motion execution. Each tentacle j is a semicircle that starts in R, is tangent to X, and is characterized by its curvature (i.e., inverse radius) κ_j, which belongs to $\mathcal{K}$, a uniformly sampled set:

$$\kappa_j \in \mathcal{K} = \{-\kappa_M, \ldots, 0, \ldots, \kappa_M\}.$$

The maximum desired curvature $\kappa_M > 0$, must be feasible considering the robot kinematics. In Fig. 2(b, c), the straight and the sharpest counterclockwise ($\kappa = \kappa_M$) tentacle are dashed. When a total of 3 tentacles is used, these correspond respectively to $j = 2$ and $j = 3$. Each tentacle j is characterized by two classification areas (*dangerous* and *collision*), which are obtained by rigidly displacing, along the tentacle, two rectangular boxes, with decreasing size, both overestimated with respect to the real robot dimensions. For each tentacle j, the sets of cells belonging to the two classification areas (shown in Fig. 2) are noted $\mathcal{D}_j$ and $\mathcal{C}_j \subset \mathcal{D}_j$. As we will show below, the largest classification area $\mathcal{D}$ will be used to select the safest tentacle, while the thinnest one $\mathcal{C}$ determines the eventual necessary deceleration.

C. Robot occupation times

For each dangerous cell in tentacle j, i.e., for each cell $\mathbf{c}_i \in \mathcal{D}_j$), we compute the *robot occupation time* t_{ij}. This is an estimate of the time at which the large box will enter the cell, assuming the robot follows the tentacle at the current velocity. To calculate t_{ij}, we assume that the robot motion is uniform, and displace the box at the current robot linear velocity v, and at angular velocity $\omega_j = \kappa_j v$. We can then calculate robot occupation time t_{ij}:

$$t_{ij}(\mathbf{c}_i, v, \kappa_j) \in I\!R^{+}.$$

For instance, if the robot is not moving ($v = 0$), for every tentacle j, the cells on the box will have $t_{ij} = 0$, and all other cells in $\mathcal{D}_j$ will have $t_{ij} = \infty$. Also note that for a given cell, t_i may differ according to the tentacle that is considered. In Fig. 2(b, c), the cells $\mathbf{c}_i \in \mathcal{D}_j$ have been displayed in grey, increasingly light with increasing t_{ij}.

D. Dangerous and collision instants

Once the obstacle and robot occupation times have been calculated for each cell, we can derive the earliest time instant at which a collision between obstacle and robot may occur on each tentacle j. By either checking all cells in $\mathcal{D}_j$, or focusing just on $\mathcal{C}_j$, we discern between *dangerous instants* and *collision instants*. These are defined as:

$$t_j = \inf_{\mathbf{c}_i \in \mathcal{D}_j} \{t_{ij} : t_{i0} \leq t_{ij} \leq t_{if}\},$$

and

$$t_j^c = \inf_{\mathbf{c}_i \in \mathcal{C}_j} \{t_{ij} : t_{i0} \leq t_{ij} \leq t_{if}\}.$$

In each case, we seek the minimum robot occupation time among the cells that can be simultaneously occupied by both obstacle and box. Assuming constant robot and obstacle velocities, these metrics give a conservative approximation of the time that the robot can travel along the tentacle without colliding. Obviously, overestimating the bounding boxes size leads also to more conservative values of t_j and t_j^c. In the following, we explain how these metrics are used: in particular, with t_j we assess the danger on each tentacle to

decide whether to follow it or not, while t_j^c determines if the robot should decelerate on tentacle j. Computation of t_j and t_j^c is illustrated, for $j = \{2, 3\}$, in the example of Fig. 2.

E. Tentacle risk function

The danger on each tentacle is assessed by *tentacle risk function* H_j. This scalar function is derived from the tentacle dangerous instant, and will be used by the controller as explained in Sect. V. We use t_j and tuned thresholds $t_d > 0$ and $t_s > t_d$ (d stands for dangerous, and s for safe), to design the tentacle risk function:

$$H_j = \begin{cases} 0 & \text{if } t_j \geq t_s \\ \frac{1}{2}\left[1 + \tanh\left(\frac{1}{t_j - t_d} + \frac{1}{t_j - t_s}\right)\right] & \text{if } t_d < t_j < t_s \\ 1 & \text{if } t_j \leq t_d. \end{cases}$$

Note that H_j smoothly varies from 0, when possible collisions are in the far future, to 1, when they are forthcoming. If $H_j = 0$, the tentacle is tagged as *clear*. All the H_j are compared (with a strategy explained below), to determine H in (3) and select the *best tentacle* for navigation.

V. CONTROL SCHEME

In our control scheme, the desired behaviour of the robot is related to the surrounding obstacles. When the environment is safe, the vehicle should progress forward while remaining near the taught path, with camera pointing forward ($\varphi = 0$). If avoidable obstacles are present, we apply a robot rotation for circumnavigation with an opposite camera rotation to maintain visibility. The rotation makes the robot follow the *best* tentacle in $\mathcal{K}$, which is selected using the strategy explained below. Finally, if collision is inevitable, the vehicle should simply stop. To assess the danger at time t, we use *situation risk function* $H \in [0, 1]$, also defined below.

Stability of the desired tasks has been guaranteed in [1] by:

$$\begin{cases} v = (1 - H)\, v_s + H v_u \\ \omega = (1 - H)\, \frac{\lambda_x (x^* - x) - j_v v_s + \lambda_\varphi j_{\dot\varphi} \varphi}{j_\omega} + H \kappa_b v_u \\ \dot\varphi = H \frac{\lambda_x (x^* - x) - (j_v + j_\omega \kappa_b) v_u}{j_{\dot\varphi}} - (1 - H)\, \lambda_\varphi \varphi \end{cases} \quad (3)$$

In the above equations:

- H is the risk function on the best tentacle: $H = H_b$; hence, it is null if and only if the best tentacle is clear.
- $v_s > 0$ is the translational velocity in the *safe context* (i.e., when $H = 0$). It must be maximal on straight path portions, and smoothly decrease when the features quickly move in the image, i.e., at sharp robot turns (large ω), and when the camera pan angle φ is strong. The expression of $v_s(\omega, \varphi)$ is given in [1].
- $v_u \in [0, v_s]$ is the translational velocity in the *unsafe context* ($H = 1$). It is designed as:

$$v_u(\delta_b) = \begin{cases} v_s & \text{if } t_b^c \geq t_s^c \\ v_s \sqrt{t_b^c - t_d^c / t_s^c - t_d^c} & \text{if } t_d^c < t_b^c < t_s^c \\ 0 & \text{if } t_b^c \leq t_d^c \end{cases}$$

  (with $t_d^c > 0$ and $t_s^c > t_d^c$ two thresholds corresponding to dangerous and safe collision times) to guarantee that the vehicle decelerates (and eventually stops) as the collision instant on the best tentacle t_b^c decreases.
- x and x^* are abscissas of the feature centroid respectively in the current and next key image.
- $\lambda_x > 0$ and $\lambda_\varphi > 0$ are empirical gains determining the convergence trend of x to x^* and of φ to 0.
- j_v, j_ω and $j_{\dot\varphi}$ are the components of the Jacobian relating $\dot x$ and $\mathbf{u}$. Their expression is given in [1].
- κ_b is the curvature of the best tentacle. Here we detail how such tentacle is determined. Initially, we calculate the path curvature that the robot would follow if $H = 0$:

$$\kappa = \omega / v = \left[\lambda_x (x^* - x) - j_v v_s + \lambda_\varphi j_{\dot\varphi} \varphi\right] / j_\omega v_s.$$

  In [1], we proved that κ is always well-defined, i.e., that $j_\omega \neq 0$. We constrain κ to the interval of feasible curvatures $[-\kappa_M, \kappa_M]$, and derive its two neighbors in $\mathcal{K}$: κ_n and κ_{nn}. Let κ_n be the nearest one, denoted as the *visual tentacle*[1]. Its situation risk function H_v is obtained by linear interpolation of the neighbours:

$$H_v = \frac{(H_{nn} - H_n)\, \kappa + H_n \kappa_{nn} - H_{nn} \kappa_n}{\kappa_{nn} - \kappa_n}.$$

  If $H_v = 0$, the visual tentacle is clear and can be followed: we set $\kappa_b = \kappa_n$. Instead, if $H_v \neq 0$, we seek a clear tentacle ($H_j = 0$). First, we search among the tentacles between the visual task one and the best one at the previous iteration[2], noted κ_{pb}. If many are present, the closest to the visual tentacle is chosen. If none of the tentacles within $[\kappa_n, \kappa_{pb}]$ is clear, we search among the others. If no tentacle in $\mathcal{K}$ is clear, the one with minimum H_j is chosen. Ambiguities are again solved first with the distance from κ_n, then from κ_{nn}.

Let us shortly recall the main features of (3), which are detailed in [1]. When $H = 0$ (i.e., if the 2 neighbour tentacles are clear), the robot tracks at its best the taught path: the image error is regulated by ω, while v is set to v_s to improve tracking, and the camera is driven forward ($\varphi = 0$). When $H = 1$, $\dot\varphi$ ensures the visual task, and the two other inputs guarantee that the best tentacle is followed: $\omega / v = \kappa_b$. In general ($H \in [0, 1]$), the robot navigates between the taught and the best paths, and a high velocity v_s can be applied if the path is clear for future time t_s^c.

VI. EXPERIMENTS

Here, we report the simulated and real experiments (also shown in the video attached to this paper) that we performed to validate our approach. We compare the new approach that is presented here, and that takes into account the obstacle velocities, with the original one designed in [1]. In the following, we denote these respectively as approach M and S (for Mobile and Static). All experiments have been carried out on our CyCab vehicle, set in car-like mode (i.e., using the front wheels for steering). For simulations, we made use of Webots[3], where we designed a virtual CyCab, and distributed random visual features, represented by spheres, in the environment. The CyCab is equipped with a coarsely calibrated 640×480 pixels $70°$ field of view, B&W Marlin (F-131B) camera mounted on a TRACLabs Biclops Pan/Tilt head (the tilt angle is null, to keep the optical axis parallel to the ground), and with a 4-layer, $110°$ scanning angle, laser SICK LD-MRS. The grid is built by projecting the laser readings from the 4 layers on the ground, and by using: $X_M = Y_M = 10$ m, $X_m = -2$ m, $Y_m = -10$ m. The cells have size 20×20 cm. For the situation risk function, we use $t_s = 6$ s and $t_d = 4.5$ s, for the unsafe translational velocity,

[1] We consider that intervals are defined even when the first endpoint is greater than the second: $[\kappa_n, \kappa_{nn})$ must be read $(\kappa_{nn}, \kappa_n]$ if $\kappa_n > \kappa_{nn}$.

[2] At the first iteration, we set $\kappa_{pb} = \kappa_n$.

[3] www.cyberbotics.com

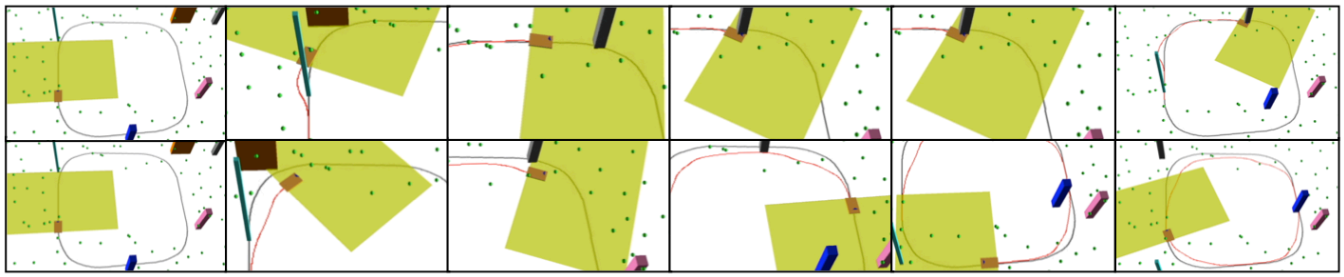

Fig. 3. Six steps of the simulations: the taught path (black) must be followed by the robot (orange) with methods S (top) and M (bottom) and 4 moving and 1 static obstacles. Visual features are represented in green, the occupancy grid in yellow, and the replayed paths in red.

Fig. 4. First real experiment: Comparison between methods S (top) and M (bottom) as a pedestrian crosses the path in front of the robot.

we use $t_s^c = 5$ s, and $t_d^c = 2$ s, and as control gains: $\lambda_x = 1$ and $\lambda_\varphi = 0.5$. A compromise between computational cost and control accuracy must be reached to tune the size of $\mathcal{K}$, i.e., its sampling interval. In all experiments, we used 21 tentacles, with $\kappa_M = 0.35$ m^{-1}.

At first, no obstacle is present in the environment, and the robot is driven along a taught path. Then, moving and static objects are present on the path, while the robot replays it to follow the key images. The metrics for assessing the experiments are the image error with respect to the visual database $x - x^*$ (in pixels), and the robot linear velocity v, both averaged over the whole experiment and denoted respectively $\bar{e}$ and $\bar{v}$. We do not consider the 3D pose error with respect to the taught path, since our task is defined in the image space, and not in the pose space. Besides, some portions of the replayed paths, corresponding to the obstacle locations, are far from the taught ones. However, these deviations are indispensable to avoid collisions.

Let us firstly describe the simulations, shown in Fig. 3. The taught visual path is a closed clockwise loop of $N = 20$ key images, and the robot must replay it, while avoiding 4 moving obstacles, with velocity norms up to 1 ms^{-1}, and a static one. Higher obstacle velocities are difficult to estimate due to the low frequency of laser processing (12.5 Hz). However, it is noteworthy to point out that 1 ms^{-1} is the walking speed of a quick pedestrian. With approach M, the vehicle is able to follow the whole path without colliding, whereas when S is used, the robot collides with the third obstacle. Let us now detail the robot behaviour in the two cases. The first obstacle (a cyan box moving straight towards the robot) is avoided by both approaches, although with M motion prediction leads to a smoother and earlier circumnavigation. With M, the robot is faster, and reaches the brown box while it is crossing its way; but since the box is expected to leave, the robot just waits for the path to return free. With S, the robot arrives at the same point late, when the box is far. The third, grey box moves straight towards the robot, like the cyan one. Since it is slightly faster, this time S is not reactive enough, and a collision occurs. On the other hand, with M the grey box as well as the remaining

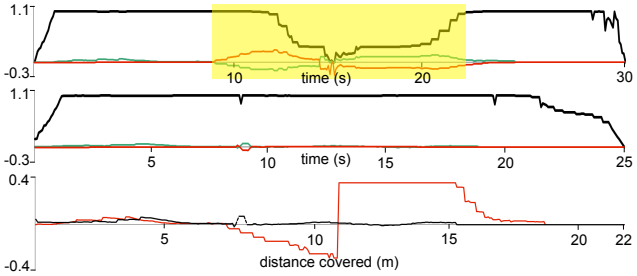

Fig. 5. First real experiment. Top and center, respectively: control inputs using S and M, with v (black, in ms^{-1}), ω (green, in rads^{-1}), $\dot{\varphi}$ (red, in rads^{-1}), and iterations with strong H highlighted in yellow. Bottom: applied curvature ω/v (in m^{-1}) using S (red) and M (black).

pink and blue ones, are easily avoided. The new approach also prevails in speed: the average velocity $\bar{v} = 0.67$ ms^{-1} with M, and $\bar{v} = 0.49$ ms^{-1} with S, although the image errors are alike ($\bar{e} = 41$ pixels with M, and $\bar{e} = 42$ with S) .

After the simulations, the framework has been ported on our CyCab vehicle. First, we have compared methods S and M in an experiment, where a pedestrian crosses the taught path in front of the robot. Then, in a second experiment, two pedestrians are passing during navigation: one crosses the path, and the other walks straight towards the robot.

The first experiment is shown in Fig. 4, with control inputs in Fig. 5. With controller S (top in both figures), the robot attempts avoidance on the right, since tentacles on the left are occupied by the person. This is clearly a doomed strategy, which leads the robot toward the pedestrian. Then, the robot must decelerate and almost stop ($v \approx 0$ after 15 s) when the pedestrian is near. Navigation is resumed only once the path is clear again. On the other hand, with controller M, as the pedestrian walks, the prediction of his future position makes him irrelevant from a safety viewpoint: risk function H (yellow in Fig. 5), which was relevant with S, is now null. Hence, the robot does not need to decelerate ($\bar{v}$ is 0.89 ms^{-1} with M, and 0.76 ms^{-1} with S) nor to deviate from the path (in Fig. 5, the applied curvature is smaller). The image error is also reduced with M: $\bar{e} = 7$ instead of 12 pixels.

For the second experiment, we show relevant iterations with the corresponding occupancy grids and currently viewed

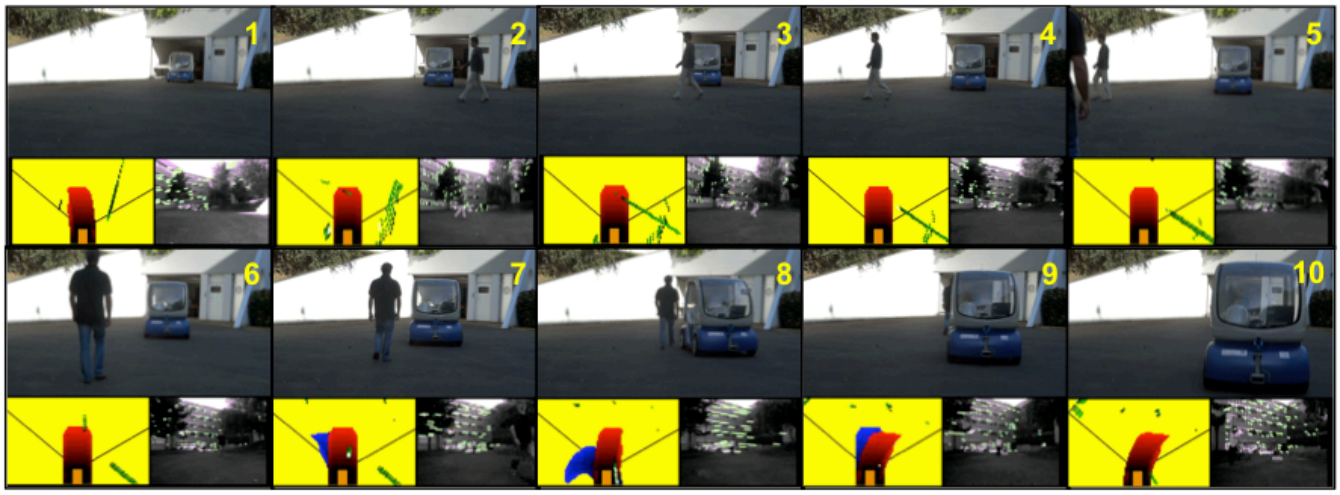

Fig. 6. Ten relevant iterations of the second experiment, where the robot avoids two pedestrians while replaying the taught path with approach M. For each iteration, we show the occupancy grid (left) and current image (right). In the occupancy grid, the dangerous cell sets associated with the visual tentacle and to the best tentacle (when different) are respectively shown in red and blue, and two black segments indicate the scanner amplitude. Only cells that we predict to be occupied in the next T s have been drawn in green. The green segments link the current and next key image points.

images, in Fig. 6. This time, the robot is controlled with M, as two pedestrians interfere with the navigation. In the occupancy grid, the propagation of cells occupied by the persons is visible at iterations 2-8. With the crossing pedestrian (iterations 2-5), since no collision is predicted, the robot keeps following the visual tentacle (red). Instead, with the forward walking pedestrian, a collision is predicted at iteration 7; then, the robot selects the best tentacle (blue) to avoid the person. Visual path replaying is again successful, with $\bar{v} = 0.87$ ms^{-1} and $\bar{e} = 10$ pixels.

VII. CONCLUSIONS

In this work, we have introduced a novel reactive approach that takes into account obstacle velocities to achieve safe visual navigation in outdoor scenarios. To estimate the obstacle velocities, we have designed a Kalman-based observer. Then, we utilize the velocities to predict possible collisions between robot and obstacles within a tentacle-based scheme. Our approach is validated in a series of experiments, where it is compared with a similar controller that does not consider obstacle velocities. We show that, by predicting the obstacle displacements within the candidate tentacles, the robot behaviour is safer and smoother, and higher velocities can be attained. In the future, we will investigate more realistic scenarios, where obstacles are not translating, as assumed here, and can approach the vehicle from behind. For the latter case, the current configuration (forward-looking lidar) must be modified.

REFERENCES

[1] A. Cherubini and F. Chaumette, "Visual navigation of a mobile robot with laser-based collision avoidance", *Int. Journal of Robotics Research*, OnlineFirst, September 4, 2012 DOI:10.1177/0278364912460413.

[2] S. Thrun, M. Montemerlo, H. Dahlkamp, D. Stavens, A. Aron, J. Diebel, P. Fong, J. Gale, M. Halpenny, G. Hoffmann, K. Lau, C. Oakley, M. Palatucci, V. Pratt, P. Stang, S. Strohband, C. Dupont, L.-E. Jendrossek, C. Koelen, C. Markey, C. Rummel, J. Van Niekerk, E. Jensen, P. Alessandrini, G. Bradski, B. Davies, S. Ettinger, A. Kaehler, A. Nefian and P. Mahoney, "Stanley: The robot that won the DARPA Grand Challenge" in *Journal of Field Robotics*, vol. 23, no. 9, 2006, pp. 661 - 692.

[3] U. Nunes, C. Laugier and M. Trivedi, "Introducing perception, planning, and navigation for Intelligent Vehicles" in *IEEE Trans. on Intelligent Transportation Systems*, vol. 10, no. 3, 2009, pp. 375–379.

[4] A. Broggi, L. Bombini, S. Cattani, P. Cerri and R. I. Fedriga, "Sensing requirements for a 13000 km intercontinental autonomous drive", *IEEE Intelligent Vehicles Symposium*, 2010, San Diego, USA.

[5] M. Buehler, K. Lagnemma and S. Singh (Editors), "Special Issue on the 2007 DARPA Urban Challenge, Part I-III", in *Journal of Field Robotics*, vol. 25, no. 8–10, 2008, pp. 423–860.

[6] J. Minguez, F. Lamiraux and J.-P. Laumond, "Motion planning and obstacle avoidance", in *Springer Handbook of Robotics*, B. Siciliano, O. Khatib (Eds.), Springer, 2008, pp. 827–852.

[7] P. Fiorini and Z. Shiller, "Motion Planning in Dynamic Environments Using Velocity Obstacles", in *Int. Journal of Robotics Research*, vol. 17, no. 7, 1998, pp. 760 – 772.

[8] D. Wilkie and J. Van den Berg, D. Manocha, "Generalized Velocity Obstacles", *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2009.

[9] A. Wu and J P. How, "Guaranteed infinite horizon avoidance of unpredictable, dynamically constrained obstacles", in *Autonomous Robots*, vol. 32, 2012, pp. 227-242.

[10] D. Fox, W. Burgard and S. Thrun, "The Dynamic Window approach to obstacle avoidance", in *IEEE Robotics and Automation Magazine*, vol. 4, no. 1, 1997, pp. 23–33.

[11] B. Damas and J. Santos-Victor, "Avoiding Moving Obstacles: the Forbidden Velocity Map", *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems*, 2009.

[12] M. Seder and I. Petrović, "Dynamic window based approach to mobile robot motion control in the presence of moving obstacles", *IEEE Int. Conf. on Robotics and Automation*, 2007.

[13] C. Fulgenzi, A. Spalanzani, C. Laugier, "Probabilistic motion planning among moving obstacles following typical motion patterns", in *IEEE/RSJ Int. Conf. on Intelligent RObots and Systems*, 2009.

[14] T. Fraichard, "A Short Paper about Motion Safety", *IEEE Int. Conf. on Robotics and Automation*, 2007.

[15] S. Bouraine, T. Fraichard and H. Salhi, "Provably safe navigation for mobile robots with limited field-of-views in dynamic environments", in *Autonomous Robots*, vol. 32, 2012, pp. 267-283.

[16] F. von Hundelshausen, M. Himmelsbach, F. Hecker, A. Mueller, and H.-J. Wuensche, "Driving with tentacles - Integral structures of sensing and motion", in *Journal of Field Robotics*, vol. 25, no. 9, 2008, pp. 640 – 673.

[17] E. Royer, M. Lhuillier, M. Dhome and J.-M. Lavest, "Monocular vision for mobile robot localization and autonomous navigation", in *Int. Journal of Computer Vision*, vol. 74, no. 3, 2007, pp. 237–260.

[18] D. Zeng Wang, I. Posner and P. Newman, "What Could Move? Finding Cars, Pedestrians and Bicyclists in 3D Laser Data", in *IEEE Int. Conf. on Robotics and Automation*, 2012.

[19] N. Wojke and M. Häselich, "Moving Vehicle Detection and Tracking in Unstructured Environments", in *IEEE Int. Conf. on Robotics and Automation*, 2012.